Lego Robotics and the Personal Computer

An Honors Thesis Project (HONRS 499)

by

Nathan Erwin

Thesis Advisor

Dr. Paul Buis

Ball State University

Muncie, Indiana

April 30, 1996

Graduation: May 4, 1996

The purpose of this project is to develop an interface in the C programming language for the Dacta Lego Robotics tools. The goal is to be able to have a set of programming tools so that future programmers could be able to write programs to utilize and control the Lego Robotics equipment.

The platform that this project was developed on is a PC running Borland C++ version 4.51. The code and header files are written in C, and they are for use in DOS based applications. By using this language, others may include the files into their C or C++ programs. This allows more flexibility in teaching and learning about robotics than the current control program provides.

Provided in the following pages is the source code that has been developed, as well as sources that this work has been based upon.

Lego Robotics Main Source

Andrew Carol
carol@edfua0.ctis.af.mil

Set up computer with 9600bps, no parity, 1 stop bit

How to start the interface box:

Send: p\0 (The letter p and a null character)

Box will then send back various copyright things.

Then send: ###Do you byte, when I knock?$$$

Box will send back: ###Just a bit off the block!$$$

Software should send "alives" (0x02) every two seconds. They need not be sent as long as commands are being issued at least every two seconds. However, it may be easier to just send alives regardless of commands being sent.

All of the commands are at least one byte in length.

All ports have 3 independent states: on/off, power level (0-8), left/right.

Port Command Notations:

mmmmmmmm	mask for up to 8 output ports, most significant bit is the eighth port, least significant bit is the first port
ppp	single input or output port in binary
nnn	power level, offset by 1 (i.e., 0 = power level 1)

Single Port Commands:

reverse	00100ppp
on	00101ppp
off	00111ppp
setleft	01000ppp
setright	01001ppp

Multiport Commands:

off	10010000 mmmmmmmm
on	10010001 mmmmmmmm
setpower 0	10010010 mmmmmmmm
setright	10010011 mmmmmmmm
setleft	10010100 mmmmmmmm
reverse	10010101 mmmmmmmm
setpower n+1	10110nnn mmmmmmmm

Sample Input (Given in hexadecimal)

Switch:

pressed in	00B8
released out	03FF

Temperature

freezing (F)	02F8
98 degrees F	1D3

Source Code

```
/*
*       dacta.h
*       Lego Dacta Control Procedures
*        Function Prototypes
*
*               Written by Nathan Erwin
*               Honors 499 - Undergraduate Thesis Independent Project
*
*
*/

#include <bios.h>
#include <conio.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*  "Invisible functions"  */
static int valid_port(int);
static int valid_input_location(char);
static int valid_output_location(char);
static int valid_power(int);

/*  "Public" Lego Functions  */
int Init_Box(int);
int Set_Motor_Power(int, char, int);
int Set_Motor_Switch(int, char, int);
int Check_Switch_Position(int, char);
```

```
/*
*       dacta.c
*       Lego Dacta Control Procedures
*
*               Written by Nathan Erwin
*               Honors 499 - Undergraduate Thesis Independent Project
*
*
*/

#include <bios.h>
#include <conio.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SETTINGS (0xE0|0x00|0x00|0x03)
#define DATA_READY      0x100
#define DTR             0x01  /*  Data Terminal Ready  */
#define RTS             0x02  /*  Ready to Send  */
#define COM1PORT        0x0000  /*  Pointer to Location of COM1 Port  */
#define COM2PORT        0x0002  /*  Pointer to Location of COM2 Port */

/*  Lego defines  */
#define PORT_A          0x00
#define PORT_B          0x02
#define PORT_C          0x04
#define PORT_D          0x08
#define PORT_E          0x10
#define PORT_F          0x20
#define PORT_G          0x40
#define PORT_H          0x80
#define OFF             0x90
#define ON              0x91
#define ZEROPOWER       0x92
#define SETRIGHT        0x93
#define SETLEFT         0x94
#define REVERSE         0x95
#define INCPOWER        0x96

static int valid_port(int port) {
/*
*       This procedure will determine if the port is legal
*
*       Input:  the port number to check
*       Output:  -1 if the port is invalid
*                0  if it is COM port 1
*                1  if it is COM port 2
*
*/

   if (port==1)
      return(port-1);
   else if (port==2)
      return(port-1);
   else
      return(-1);

}
```

```
static int valid_input_location(char loc) {
/*
*       This procedure will determine if the input location is legal
*
*       Input:  the connection point, valid inputs are A-D
*       Output:  0 if the connection is invalid
*                The port location if the connection is valid
*
*/

   switch (loc) {
      case 'A':
         return(PORT_A);
      case 'B':
         return(PORT_B);
      case 'C':
         return(PORT_C);
      case 'D':
         return(PORT_D);
      default:
         return(0);
   }
}

static int valid_output_location(char loc) {
/*
*       This procedure will determine if the input location is legal
*
*       Input:  the connection point, valid inputs are E-H
*       Output:  0 if the connection is invalid
*                1 if the connection is valid
*
*/

   switch (loc) {
      case 'E':
         return(PORT_E);
      case 'F':
         return(PORT_F);
      case 'G':
         return(PORT_G);
      case 'H':
         return(PORT_H);
      default:
         return(0);
   }
}

static int valid_power(int power) {
/*
*       This procedure will determine if the power level is valid
*
*       Input:  the power level to check, valid values are 0-3
*       Output: 0 if the power level is invalid, 1 if valid
*
*/

   if ((power >= 0) && (power <= 3))
      return(1);
```

```
   else
      return(0);

}

int Init_Box(int serial_port) {
/*
*       This procedure will initialize the connection between the
*               serial port and the Dacta Control Box.
*
*       Input:  serial port the box is connected to
*       Output:  0 if the connection failed
*                1 if the connection is successful
*/

   int stat;
   int i;
   int port;
   char *Init_Send = "###Do you byte, when I [illegible]$$$";
   char [illegible] = "###[illegible] off the bl[illegible]";
   char [illegible];
   int [illegible] = 0;
   int [illegible] = 0;
   int far *RS232_Addr;

    * Validate the port *
   port = Valid_Port(serial_port);
   if (port < 0) {
      fprintf(stderr, "Invalid port\n", port);
      return(0);
   }

    * Determine port location of the COM port we are using  */
   if (port == 0) {
      RS232_Addr = MK_FP(0x0040, COM1PORT);
      if (!*RS232_Addr) {
         fprintf(stderr, "Cannot attach RS232_Addr to COM1\n");
         return(0);
      }
   }
   else if (port == 1) {
      RS232_Addr = MK_FP(0x0040, COM2PORT);
      if (!*RS232_Addr) {
         fprintf(stderr, "Cannot attach RS232_Addr to COM2\n");
         return(0);
      }
   }
   else {
      fprintf(stderr, "Invalid port\n");
      return(0);
   }

    * Initialize the port  */
   stat = bioscom(0, SETTINGS, port);
   outportb(*RS232_Addr + 4, DTR|RTS);

    * This is where the p and null should go out
   stat = bioscom(1, 'p\0', port);
   stat = bioscom(2, recv[0], port);
   */
```

```
    if (stat & DATA_READY) {
       /*  Send the init string and wait for the box to respond  */
       for (i=1; i<=snd; i++) {
          stat = bioscom(1, Init_Send[i], port);
          if (stat & DATA_READY) {
             continue;
          }
          else {
             fprintf(stderr, "Serial port not ready to send.\n");
             return(0);
          }
       }

       /*  If we get the right init string back, we're ok to go  */
       for (i=1; i<=rcv; i++) {
          stat = bioscom(2, recv[i], port);
          if (stat & DATA_READY) {
             continue;
          }
          else {
             fprintf(stderr, "Serial port not sending.\n");
             return(0);
          }
       }

       if (!strcmp(recv, Init_Recv))
          return(1);
       else
          return(0);
       }
       else {
          fprintf(stderr, "Serial port not ready for initialization.\n");
          return(0);
       }

}

int Set_Motor_Power(int serial_port, char location, int power_level) {
/*
*       This procedure will set the speed of the motor
*
*       Input:  the port the box is connected to
*     the box location the motor is connected to
*       Output:  0 if the set failed
*                1 if the set was successful
*/

   int port;
   int loc;
   int stat;

    /*  Validate the box port  */
   if ((port=valid_port(serial_port)) < 0)
      return(0);

    /*  Validate the connection point  */
   if ((loc=valid_output_location(location)) == 0)
      return(0);
```

```
   /*  Validate the power level  */
   if (!valid_power(power_level))
      return(0);

   power_level = (power_level-1) + 0xB0;

   /*  Multiport:  We have to send 2 bytes to the port*/
   stat = bioscom(1,power_level,port);
   if (!(stat & DATA_READY))
      return(0);

   stat = bioscom(1,loc,port);
   if (stat & DATA_READY)
      return(1);
   else
      return(0);

}

int Set_Motor_Switch(int serial_port, char location, int switch_pos) {
/*
*       This procedure will turn the motor on or off
*
*       Input:  the port the motor is connected to
*               the switch position:  0 = off, 1 = on
*       Output:  0 if the switch failed
*                1 if the switch was successful
*/

   int port;
   int stat;

   /*  Validate the box port  */
   if ((port=valid_port(serial_port)) < 0)
      return(0);

   /*  Validate the connection point  */
   if (!valid_output_location(location))
      return(0);

   /*  Validate the switch position  */
   if ((switch_pos != 1) && (switch_pos != 0))
      return(0);

   stat = bioscom(1, REVERSE, port);
   if (stat & DATA_READY)
      return(1);
   else
      return(0);

}

int Check_Switch_Position(int serial_port, char location) {
/*
*       This procedure will check the status of an input switch
*
*       Input:  the port the switch is connected to
*               the switch position:  0x3FF = off, 0x0B8 = on
*       Output:  0x3FF if the switch was released out
*                0x0B8 if the switch was pushed in
```

```
*/

   int position = 0;

   /*  Validate the box port  */
   if (valid_port(serial_port) < 0)
      return(0);

   /*  Validate the connection point  */
   if (!valid_input_location(location))
      return(0);

   /*  There isn't a way to read in from the input ports right now  */
   return(position);
```

Sample Program

```
/*
*     legoctrl.c
*        This is an example program using the Dacta Lego control
*           code developed by Nathan Erwin for Honors 499
*
*/

#include "dacta.h"

void main() {

   int value = 0;
   int port = 0;
   char command[30];
   int len = 0;
   char loc[2];

   printf("Enter the port the box is connected to:  ");
   scanf("%d", &port);
   if (Init_Box(port)) {
      fprintf(stdout, "Box initialized successfully\n");
   }
   else {
      fprintf(stderr, "Error initializing box\n");
   }

   while(1) {
      /*  Give the user a prompt  */
      printf(">> ");
      scanf("%s", command);
      len = strlen(command);

      /*  This is where the commands are parsed  */
      /*  Only the motor commands are working now  */
      if (!strncmpi(command, "power", len)) {
         printf("Location:  ");
         scanf("%s", loc);
         printf("Level:  ");
         scanf("%d", &value);
         if (!Set_Motor_Power(port, loc[0], value)) {
            fprintf(stderr, "\nInvalid power level\n");
         }
      }
      else if (!strncmpi(command, "switch", len)) {
         printf("Location:  ");
         scanf("%s", loc);
         printf("Position:  ");
         scanf("%d", &value);
         if (!Set_Motor_Power(port, loc[0], value)) {
            fprintf(stderr, "\nInvalid power level\n");
         }
      }

      else if (!strncmpi(command, "help", len)) {
         printf("\nCommands:\n");
         printf("power\n");
         printf("switch\n");
         printf("exit\n\n");
      }

      else if (!strncmpi(command, "exit", len)) {
         exit(0);
      }

      else {
```

```
            fprintf(stderr, "\nInvalid command\n");
        }
    }
}
```